

Model obiektowy

Pojęcia podstawowe

- **Klasa** - abstrakcyjna reprezentacja, zawiera definicję struktury i zachowania w postaci składowych i metod
- **Obiekt** - wystąpienie klasy, reprezentacja konkretnego bytu
- **Metoda** - zdolność obiektu do wykonania czynności
- **przekazywanie komunikatów** - sposób przekazywania danych lub wywoływania czynności między obiektami
- **Dziedziczenie** - uproszczony sposób specjalizacji klas
- **Enkapsulacja** - ukrywanie złożoności obiektu za interfejsem
- **Polimorfizm** - zdolność występowania obiektów w kontekście ich klas-przodków

Zalety modelu obiektowego

- Modelowanie
 - naturalna reprezentacja złożonych obiektów świata rzeczywistego
 - naturalna reprezentacja relacji kompozycji
 - naturalna reprezentacja interakcji między obiektami
 - bezpośrednie powiązanie operacji z danymi
- Programowanie
 - wielokrotne użycie kodu
 - modularność i łatwość pielęgnacji
 - zwiększone bezpieczeństwo i elastyczność
 - brak konieczności konwersji między modelami danych na styku baza danych – aplikacja

Model obiektowo-relacyjny

Model obiektowy:

- łatwość modelowania
- składowanie danych w takiej samej postaci jak w aplikacji
- związanie operacji z danymi

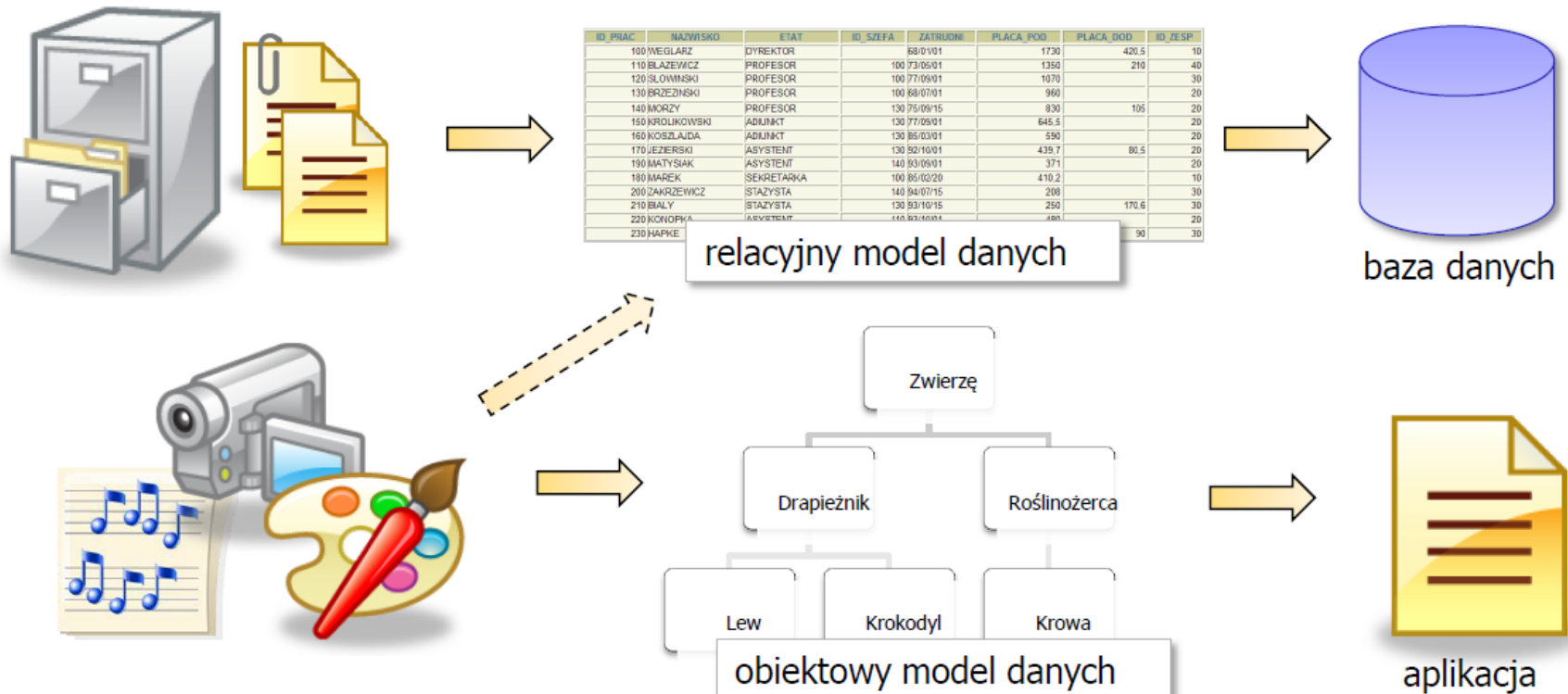
Model relacyjny:

- wysoka współbieżność
- zapytania w SQL
- kopie zapasowe i odtwarzanie po awarii
-



Model obiektowo - relacyjny

Obiektowo-relacyjny system ORACLE



Model obiektowo-relacyjny

- Umożliwia wykorzystanie mechanizmów obiektowych wewnątrz relacyjnej bazy danych
 - abstrakcyjne typy danych definiowane przez użytkownika
 - predefiniowane typy obiektowe dostarczane przez producentów oprogramowania (multimedia, dane przestrzenne)
 - dziedziczenie, enkapsulacja, polimorfizm, klasy abstrakcyjne
 - dostępne mechanizmy relacyjne - SQL, przetwarzanie transakcyjne, zarządzanie współbieżnością, ścieżki dostępu, kopie bezpieczeństwa i protokoły odtwarzania spójności
- Alternatywa dla systemów odwzorowania obiektowo-relacyjnego (ORM)

Własności obiektowo-relacyjne w ORACLE

- **Obiektowe typy danych użytkownika**
 - metody składowe, konstruktory, przeciążanie
- **Współdzielenie obiektów**
 - referencje, nawigacja
- **Dziedziczenie**
 - polimorfizm, przesłanianie metod, typy abstrakcyjne
- **Kolekcje**
 - tabele o zmiennym rozmiarze, tabele zagnieżdżone
- **Perspektywy obiektowe**

Obiekty w bazie ORACLE

- Oracle jest relacyjną bazą danych posiadającą cechy modelu obiektowego
- Obiekty w bazie danych tworzy się na podstawie wcześniej utworzonych typów obiektowych
- Każdy typ obiektowy może składać się z:
 - nazwy typu obiektowego
 - atrybutów, opisujących strukturę obiektu
 - metod będących funkcjami lub procedurami, przez które wykonywane są operacje związane z danym obiektem

Obiekty w bazie ORACLE

- Typy obiektowe i ich ciała można tworzyć jako typy danych języka SQL
- Typów obiektowych można używać w języku SQL na cztery sposoby:
 - Jako typu danych kolumn w definicjach tabel
 - Jako typu danych atrybutu obiektu przy deklarowaniu typu obiektowego
 - Jako typu danych parametru formalnego w sygnaturach funkcji i procedur
 - Jako typu wartości zwracanej przez funkcję

Obiekty w bazie ORACLE

- Obiekty w ORACLE są:
 - trwałe
 - niezależne
 - zagnieżdżone
 - Tymczasowe
- Nazwy typów obiektowych w PL/SQL muszą rozpoczynać się od litery i mogą składać się wyłącznie z liter, cyfr i podkreślenia
- Nazwy obiektów znajdują się w tej samej przestrzeni nazw, co nazwy wszystkich innych elementów bazy danych oprócz wyzwalaczy

Obiekty w bazie ORACLE

Definicja typu obiektowego składa się z dwóch części:

- specyfikacji
- ciała

```
CREATE [OR REPLACE] TYPE nazwa_typu  
AS OBJECT (definicja atrybutu [, definicja atrybutu]  
[MEMBER FUNCTION nazwa_funkcji klauzula_zwrotu_funkcji]);
```

```
CREATE [OR REPLACE] TYPE nazwa_typu  
AS OBJECT (definicja atrybutu [, definicja atrybutu]  
[MEMBER PROCEDURE nazwa_procedury]);
```

Obiekty w bazie ORACLE

- Każdy typ obiektowy składa się z dwóch części:
 - **deklaracji** (interfejsu): zbioru składowych (pól i sygnatur metod)
 - **definicji** (ciała): implementacji metod



```
CREATE TYPE Pracownik AS OBJECT (  
    nazwisko VARCHAR2(20),  
    pensja    NUMBER(6,2),  
    etat      VARCHAR2(15),  
    data_ur   DATE  
);
```

Obiekty w bazie ORACLE

- Obiekty składuje się trwale jako:
 - obiekty krotkowe
 - obiekty atrybutowe
- Do przechowywania obiektów krotkowych wykorzystuje się tabele obiektów
- Obiekty atrybutowe są przechowywane jako wartości atrybutu w zwyczajnej tabeli

Obiekty krotkowe

- Utworzenie tabeli obiektów

```
CREATE TABLE PracownicyObjTab  
OF Pracownik;
```

- Wstawienie obiektu krotkowego

```
INSERT INTO PracownicyObjTab VALUES  
(NEW Pracownik('Kowalski',2500,'ASYSTENT',  
DATE '1965-07-01'));
```

- Dostęp obiektowy i relacyjny do obiektów krotkowych

```
SELECT VALUE(p) FROM PracownicyObjTab p;  
  
SELECT * FROM PracownicyObjTab;
```

Obiekty atrybutowe

- Utworzenie tabeli z obiektami atrybutowymi

```
CREATE TABLE ProjektyTab (  
    symbol CHAR(6), nazwa VARCHAR(100),  
    budzet NUMBER, kierownik Pracownik);
```

- Wstawienie obiektu atrybutowego

```
INSERT INTO ProjektyTab VALUES  
('AB 001', 'Projekt X', 20000,  
    NEW Pracownik('Nowak', 3200, 'ADIUNKT', null));
```

- Dostęp obiektów atrybutowych i ich składowych

```
SELECT nazwa, kierownik FROM ProjektyTab;  
  
SELECT p.kierownik.nazwisko FROM ProjektyTab p;
```

alias jest wymagany w przypadku dostępu do składowych obiektu atrybutowego

Definiowanie metod typu obiektowego

- Obiekty w bazie danych tworzy się na podstawie wcześniej utworzonych typów obiektowych
- Każdy typ obiektowy może składać się z:
 - Nazwy typu obiektowego
 - Atrybutów, opisujących strukturę obiektu
 - Metod będących funkcjami lub procedurami, przez które wykonywane są operacje związane z danym obiektem.

Tworzenie obiektów

- Do tworzenia obiektu służy specjalna metoda – **konstruktor**
- Każdy typ obiektowy posiada **konstruktor atrybutowy**
 - lista argumentów zgodna z listą atrybutów typu obiektowego
 - podczas wywołania należy podać wartości wszystkich atrybutów
- Nazwa konstruktora jest identyczna z nazwą typu obiektowego
- Wywołanie konstruktora poprzedzamy słowem **NEW**
- Użytkownik może deklarować własne konstruktory
- Sposób tworzenia obiektu jest identyczny dla obiektów krotkowych i obiektów atrybutowych

Definiowanie metod typu obiektowego

- Typ obiektowy może posiadać następujące rodzaje metod
 - **MEMBER**: metody wołane na rzecz konkretnego obiektu
 - **STATIC**: metody wołane na rzecz całego typu obiektowego
 - **MAP/ORDER**: metody wykorzystywane do porównywania obiektów
 - **CONSTRUCTOR**: metody tworzące nowe obiekty
- Metoda jest definiowana dwukrotnie
 - w deklaracji typu: **sygnatura metody** (nazwa i lista argumentów)
 - w definicji typu: **ciało metody** (implementacja, kod źródłowy)

Definiowanie metod typu obiektowego

```
ALTER TYPE Pracownik REPLACE AS OBJECT (  
    nazwisko VARCHAR2(20),  
    pensja    NUMBER(6,2),  
    etat      VARCHAR2(15),  
    data_ur   DATE,  
    MEMBER FUNCTION wiek RETURN NUMBER,  
    MEMBER PROCEDURE podwyzka(p_kwota NUMBER) );
```

Definiowanie metod typu obiektowego

```
CREATE OR REPLACE TYPE BODY Pracownik AS  
  MEMBER FUNCTION wiek RETURN NUMBER IS  
  BEGIN  
    RETURN EXTRACT (YEAR FROM CURRENT_DATE) -  
           EXTRACT (YEAR FROM data_ur);  
  END wiek;  
  
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS  
  BEGIN  
    pensja := pensja + p_kwota;  
  END podwyzka;  
END;
```

Usuwanie typu obiektowego

- Raz utworzony typ obiektowy można usunąć za pomocą polecenia

`DROP TYPE nazwa_typu [FORCE]`

-
- Słowo kluczowe `FORCE` oznacza, że typ obiektowy ma zostać usunięty niezależnie od jakichkolwiek przeciwwskazań.

Typy obiektowe

- Informacje o typach obiektowych znajdują się w perspektywie słownika danych o nazwie: **USER_OBJECTS**
- Informacje o metodach typów obiektowych znajdują się odpowiednio w perspektywach słownika danych:
 - **USER_METHOD_PARAMS**
 - **USER_METHOD_RESULTS**
 - **USER_TYPE_METHODS**

Uruchamianie metod składowych

- Metody składowe mogą być uruchamiane:
 - z poziomu SQL (tylko metody będące funkcjami)
 - z poziomu PL/SQL (wszystkie metody)

```
SELECT p.nazwisko, p.data_ur, p.wiek() FROM PracownicyObjTab p;
```

```
DECLARE  
  l_kierownik Pracownik;  
BEGIN  
  SELECT kierownik INTO l_kierownik  
  FROM ProjektyTab WHERE symbol = 'AB 001';  
  
  l_kierownik.podwyzka(200);  
  
  UPDATE ProjektyTab  
  SET kierownik = l_kierownik  
  WHERE symbol = 'AB 001';  
END;
```

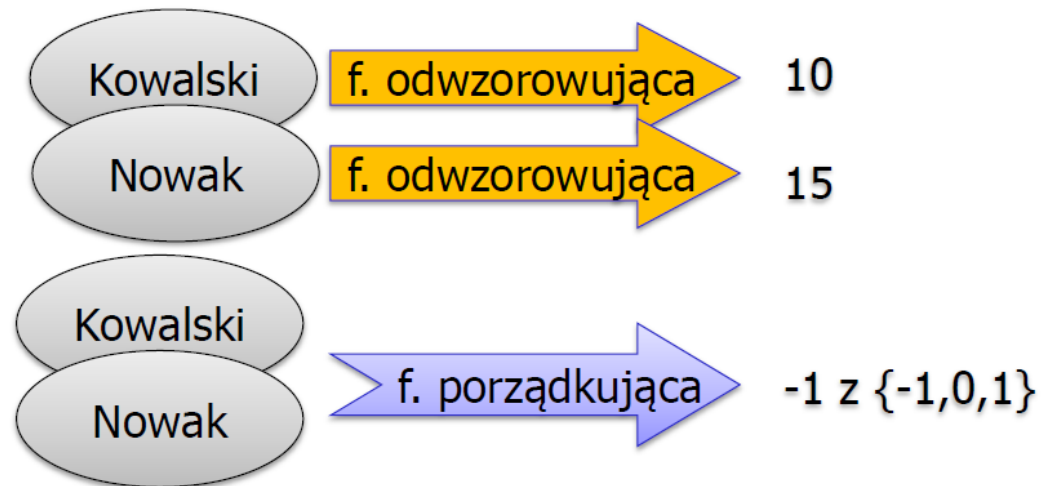
Tożsamość obiektów

- Tożsamość obiektu to unikalny i niezmienny identyfikator związany z danym obiektem przez cały cykl jego życia
 - tożsamość posiadają tylko obiekty krotkowe
 - tożsamość jest zapewniana przez OID (ang. *object identifier*)
 - identyfikatory są lokalnie i globalnie unikalne, automatycznie indeksowane i zajmują 16B; można również wykorzystać klucz podstawowy tabeli obiektowej do generowania identyfikatorów OID
 - aplikacja może wykorzystywać referencje do identyfikatorów, odczytywane za pomocą funkcji **REF ()**

```
SELECT VALUE (p) , REF (p) FROM PracownicyObjTab p;
```

Porównywanie wartości obiektów

- Wartością obiektu jest zbiór wartości jego składowych
 - dwa obiekty są równe gdy mają te same wartości składowych
 - standardowo, operatory `=` i `!=` umożliwiają porównywanie obiektów
- Porównywanie obiektów najczęściej wymaga sortowania
 - operatory relacyjne: `<`, `>`, `<=`, `>=`,
 - operator `BETWEEN ... AND ...`,
 - klauzule `ORDER BY`, `GROUP BY`, `DISTINCT`
 - porównywanie za pomocą funkcji odwzorowującej lub porządkującej



Wykorzystanie metod porównujących

- typ obiektowy może mieć tylko jedną metodę porównującą (porządkującą lub odwzorowującą)
- metody odwzorowujące są bardziej efektywne, metody porządkujące są bardziej elastyczne
- metody porównujące mogą być wywoływane jawnie lub niejawnie

```
INSERT INTO PracownicyObjTab VALUES
  (NEW Pracownik('Nowak',2000,'ADIUNKT',DATE '1961-02-15'));
INSERT INTO PracownicyObjTab VALUES
  (NEW Pracownik('Janiak',1800,'ASYSTENT',DATE '1973-12-02'));

SELECT p.nazwisko, p.pensja, p.wiek()
FROM PracownicyObjTab p
ORDER BY VALUE(p);

SELECT * FROM PracownicyObjTab p
WHERE VALUE(p) > ( SELECT VALUE(r) FROM PracownicyObjTab r
                  WHERE r.nazwisko = 'Janiak' );
```

Dodanie metody do istniejącego typu obiektowego

Dodanie sygnatury metody do deklaracji typu obiektowego

```
ALTER TYPE Pracownik ADD MAP MEMBER FUNCTION odwzoruj  
RETURN NUMBER CASCADE INCLUDING TABLE DATA;
```

Dodanie implementacji metody do definicji typu obiektowego

```
CREATE OR REPLACE TYPE BODY Pracownik AS  
  MEMBER FUNCTION wiek RETURN NUMBER IS  
  BEGIN  
    RETURN EXTRACT (YEAR FROM CURRENT_DATE) -  
           EXTRACT (YEAR FROM data_ur);  
  END wiek;  
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS  
  BEGIN  
    pensja := pensja + p_kwota;  
  END podwyzka;  
  MAP MEMBER FUNCTION odwzoruj RETURN NUMBER IS  
  BEGIN  
    RETURN ROUND(pensja,-3) + wiek();  
  END odwzoruj;  
END;
```

Konstruktor – specjalna metoda tworząca nowe obiekty

- nazwa konstruktora jest taka sama jak nazwa typu obiektowego
- użytkownik może definiować własne konstruktory, może także przesłonić domyślny konstruktor atrybutowy
- konstruktory ułatwiają inicjalizację i umożliwiają ewolucję obiektów

```
ALTER TYPE Pracownik REPLACE AS OBJECT (  
    nazwisko VARCHAR2(20),  
    pensja    NUMBER(6,2),  
    etat      VARCHAR2(15),  
    data_ur   DATE,  
    MEMBER FUNCTION wiek RETURN NUMBER,  
    MEMBER PROCEDURE podwyżka(p_kwota NUMBER),  
    MAP MEMBER FUNCTION odwzoruj RETURN NUMBER,  
    CONSTRUCTOR FUNCTION Pracownik(p_nazwisko VARCHAR2)  
        RETURN SELF AS RESULT );
```

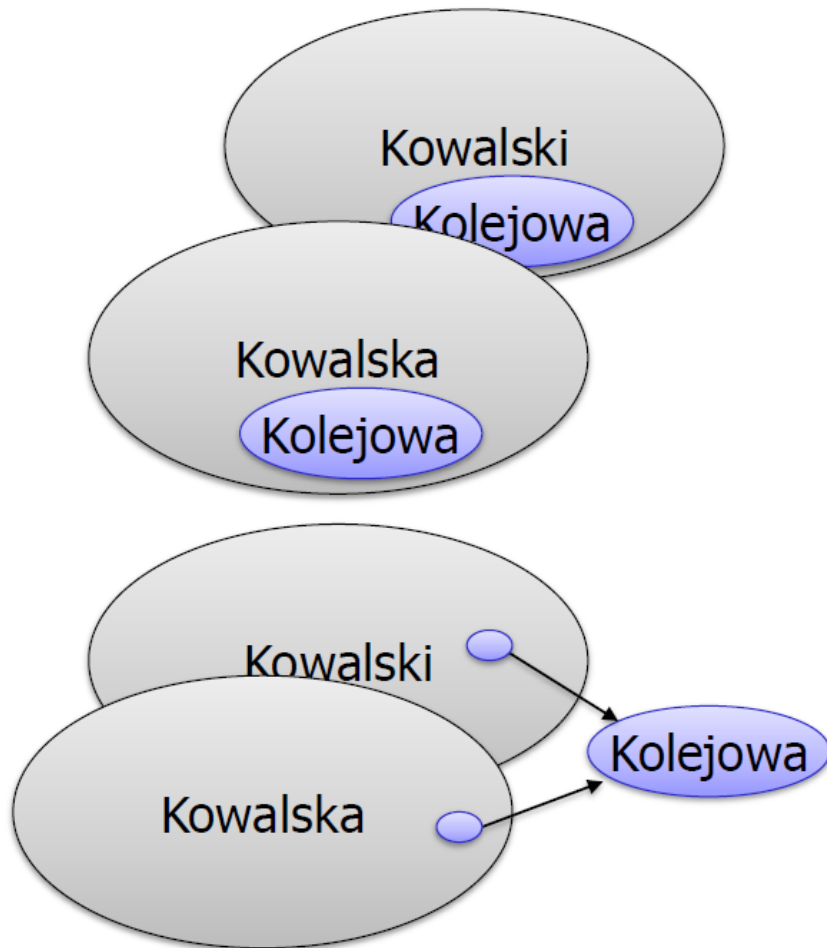
Konstruktor – implementacja

```
CREATE OR REPLACE TYPE BODY Pracownik AS  
  MEMBER FUNCTION wiek RETURN NUMBER IS  
  BEGIN  
    RETURN EXTRACT (YEAR FROM CURRENT_DATE) -  
      EXTRACT (YEAR FROM data_ur);  
  END wiek;  
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS  
  BEGIN  
    pensja := pensja + p_kwota;  
  END podwyzka;  
  MAP MEMBER FUNCTION odwzoruj RETURN NUMBER IS  
  BEGIN  
    RETURN ROUND(pensja,-3) + wiek();  
  END odwzoruj;  
  
  CONSTRUCTOR FUNCTION Pracownik(p_nazwisko VARCHAR2)  
    RETURN SELF AS RESULT IS  
  BEGIN  
    SELF.nazwisko := p_nazwisko; SELF.pensja := 1000;  
    SELF.etat := null; SELF.data_ur := null;  
    RETURN;  
  END;  
END;
```

Konstruktor – specjalna metoda tworząca nowe obiekty

- Typy obiektowe zwykle udostępniają konstruktor domyślny.
- Takie konstruktory zazwyczaj nie mają parametrów formalnych.
- W obiektach, których egzemplarze wymagają parametrów formalnych, konstruktor domyślny wywołuje konstruktor z parametrem domyślnym
- Odbywa się to w czterech etapach:
 1. Najpierw należy przygotować zmienną lokalną typu obiektowego
 2. Następnie trzeba utworzyć lokalny (wewnętrzny) egzemplarz klasy za pomocą domyślnej wartości argumentu
 3. Trzeci krok to przypisanie tymczasowego obiektu lokalnego do egzemplarza klasy
 4. Czwarty etap polega na zwróceniu uchwytu do obiektu

Współdzielenie i zagnieżdżanie obiektów



```
CREATE TYPE TAdres AS OBJECT (  
    ulica VARCHAR2(15),  
    dom NUMBER(4),  
    mieszkanie NUMBER(3));
```

```
CREATE TYPE Osoba AS OBJECT (  
    nazwisko VARCHAR2(20),  
    imie VARCHAR2(15),  
    adres TAdres);
```

```
CREATE TYPE Osoba AS OBJECT (  
    nazwisko VARCHAR2(20),  
    imie VARCHAR2(15),  
    adres REF TAdres);
```

Współdzielenie i zagnieżdżanie obiektów

- Utworzenie tabel obiektowych

```
CREATE TABLE AdresyObjTab OF TAdres;  
CREATE TABLE OsobyObjTab OF Osoba;
```

```
ALTER TABLE OsobyObjTab ADD  
SCOPE FOR(adres) IS AdresyObjTab;
```

- Wstawienie obiektów


```
INSERT INTO AdresyObjTab VALUES  
(NEW TAdres('Kolejowa',2,18));
```

```
INSERT INTO OsobyObjTab VALUES  
(NEW Osoba('Kowalska','Anna',null));
```

```
INSERT INTO OsobyObjTab VALUES  
(NEW Osoba('Kowalski','Jan',null));
```

```
UPDATE OsobyObjTab o  
SET o.adres = (  
    SELECT REF(a) FROM AdresyObjTab a  
    WHERE a.ulica = 'Kolejowa' );
```

aby zawęzić zakres referencji tabela
OsobyObjTab musi być pusta



Referencje – sposoby wykorzystania

- nawigacja jawna przez wywołanie funkcji **DEREF** ()
- nawigacja niejawna przez notację kropkową
- operacja połączenia tabel przez porównanie referencji

```
SELECT o.imie, o.nazwisko, DEREF(o.adres)  
FROM OsobyObjTab o;
```

```
SELECT o.imie, o.nazwisko, o.adres.ulica, o.adres.dom  
FROM OsobyObjTab o;
```

```
SELECT o.imie, o.nazwisko, a.ulica, a.dom, a.mieszkanie  
FROM OsobyObjTab o JOIN AdresyObjTab a  
    ON o.adres = REF(a);
```

Ograniczenia integralnościowe

- Definiowanie ograniczeń integralnościowych jest możliwe tylko dla tabel obiektowych

```
ALTER TABLE OsobyObjTab  
ADD PRIMARY KEY(imie,nazwisko);
```

```
ALTER TABLE PracownicyObjTab  
ADD CONSTRAINT niezerowa_pensja CHECK (pensja > 0);
```

```
ALTER TABLE PracownicyObjTab  
ADD CONSTRAINT niepusty_etat etat NOT NULL;
```

Ograniczenia integralnościowe

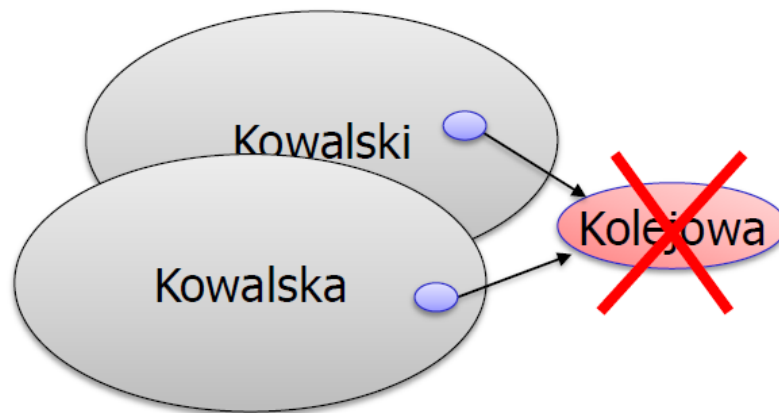
- Perspektywy **USER_TYPES** i **USER_SOURCE** słownika

```
SELECT type_name,typecode,attributes,methods FROM user_types;
```

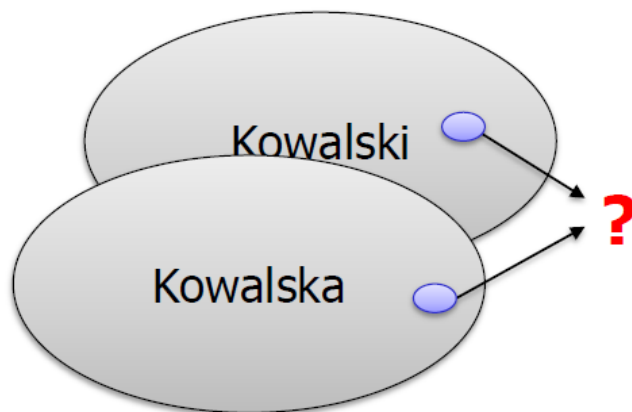
```
SELECT text FROM user_source WHERE name = 'OSOBA';
```

Referencje – sposoby wykorzystania

- Usunięcie obiektu nie usuwa referencji do obiektu



```
DELETE FROM AdresyObjTab a  
WHERE a.ulica = 'Kolejowa';
```



```
SELECT * FROM OsobyObjTab o  
WHERE o.adres IS NULL;
```

```
SELECT * FROM OsobyObjTab o  
WHERE o.adres IS Dangling;
```